ELSEVIER

Theoretical Computer Science 183 (1997) 229–251

Theoretical Computer Science

Petri nets, traces, and local model checking[1]

Allan Cheng *

BRICS,[2] Computer Science Department, University of Aarhus, Ny Munkegade, DK-8000 Aarhus C, Denmark

Abstract

It has been observed that representing concurrent behaviour as sequences of interleaved events is not satisfactory – not all sequences should be considered as likely behaviours. Taking progress fairness assumptions into account one obtains a more realistic behavioural view of concurrent systems. In this paper we consider the problem of performing model-checking relative to this behavioural view. We present a CTL-like logic which is interpreted over labelled 1-safe nets. It turns out that Mazurkiewicz trace theory provides a natural setting in which progress fairness assumptions can be formalised. We provide the first, to our knowledge, set of sound and complete tableau rules for a CTL-like logic interpreted under progress fairness assumptions.

Keywords: Fair progress; Labelled 1-safe nets; Model-checking; Maximal traces; Partial orders; Inevitability

1. Introduction

Although Petri introduced his model of concurrent systems in the early sixties [29], it has taken the community some time to focus the attention on behavioural views of concurrent systems in which concurrency or parallelism is represented explicitly [30, 19, 38, 32, 39]. This has been done by imposing more structure on models for concurrent systems – in our case, an independence relation on the transitions of labelled 1-safe nets.

As an example, consider the process agent $fix(X = a.X)|(b.c.\mathbf{0})$. Its transition graph is given below. The initial state is i and s_1 and s_2 are the only other reachable states. The agent can also be represented by the labelled 1-safe net (see Fig. 1), containing three transitions labelled a, b, and c, respectively.

The net gives us a more concrete model of the process agent. It shows that the transition labelled a is independent of those labelled b and c (Fig. 2). We can therefore add

* E-mail: acheng@daimi.aau.dk.

[1] This work has been supported by The Danish Research Councils and The Danish Research Academy.

[2] **B**asic **R**esearch in **C**omputer **S**cience, Centre of the Danish National Research Foundation.

PII S0304-3975(96)00326-X

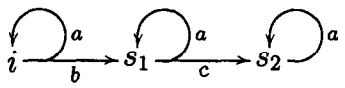


Fig. 1. Transition system for the process agent $fix(X = a.X)|(b.c.\mathbf{0})$.

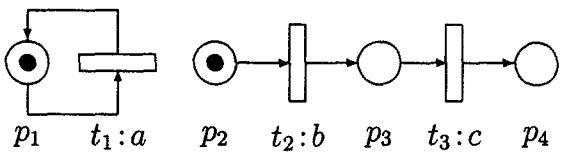


Fig. 2. Labelled 1-safe net.

more structure to the above transition system by providing a relation which explicitly states this *independence*. The new transition system is an example of a *labelled asynchronous transition system* [39]. In fact, Mukund and Nielsen [23] have shown that it is possible to obtain elementary labelled asynchronous transition systems from process agents, like the above, by introducing locations in the structural operational semantics rules for CCS.

The study of partial order semantics/"true concurrency" has developed numerous new models, e.g., concurrent and asynchronous transition system and event structures [31, 2, 32, 25, 37, 24, 38, 39]. For an overview of the relation between many of the existing models, see [39]. Common to these models is that they represent concurrency explicitly by either an independence relation (asynchronous transition system) or a conflict relation (event structures).

Our main objective is to explore the use of the extra structure of independence in the context of specification logics. Based on an independence relation on transitions (given by disjointness of the neighbourhoods of the transitions) and a generalisation of traces which takes infinite firing sequences into account, we examine a partial order semantics for labelled 1-safe nets. This semantics captures – in a formal sense – the notion of fair progress among independent event; we can then formally define which firing sequences are progress fair. We then introduce a CTL-like branching time temporal logic, P-CTL, which contains one important feature: the model-theoretic incorporation of progress. P-CTL-formulas are interpreted relative to the progress fair computations rather than all computations, as is the case for the standard interpretation of CTL. Our interpretation is conservative in the sense that it coincides with the standard CTL interpretation if the labelled 1-safe nets we consider do not exhibit concurrent behaviour. As an example, the formula $\mathsf{Ev}(\langle c\rangle tt)$ – to be read as "eventually a c-labelled transition/action is enabled" – is true of the process agent example under the assumption of progress (our interpretation), but not without (standard CTL interpretation). In process algebraic terms, our notion of fair progress – progress of independent events – intuitively corresponds to a progress fair "parallel operator".

In the standard setting of Kripke structures, model-checking of CTL-like logics has been described in [8] using a state-based algorithm and in [16, 33] using tableaux rules.

We give both a state-labelled-based method and a tableau-based method for model-checking P-CTL. These methods are both based on state-space exploration. However, they differ in the way the exploration is performed. State labelling methods explore the entire state space, labelling the states in a bottom-up fashion with the subformulas (of a given formula) they satisfy. Tableau-based methods, on the other hand, are usually referred to as "local model-checking"; the way one establishes that a state satisfies a given formula is from the given state to explore the state-space according the tableau rules. These rules typically infer the properties of a state in terms of the properties of its neighbouring states.

Our methods are conservative extensions of the existing standard methods in the sense that our methods are equivalent if the systems we consider do not exhibit concurrent behaviour.

We also determine the computational complexity of model-checking our new logic. Our results show that there is now significant computational penalty, when going from CTL to P-CTL.

In Section 2 we give the necessary basic definitions. Then, in Section 3, we introduce the logic P-CTL. Section 4 presents a global model-checking algorithm for P-CTL. Section 5 contains our main result, a set of sound and complete tableau rules for P-CTL, and, finally, in Section 6 we draw some conclusions and give directions for future research.

2. Definitions

2.1. Traces

In this section we recall some basic definitions. We start by defining *concurrent alphabets*, the fundamental structure in Mazurkiewicz trace theory [19, 20].

Definition 1 (*Concurrent alphabet and traces*).

- A *concurrent alphabet* (A, I) consists of a finite set A (the alphabet) and a symmetric and irreflexive relation $I \subseteq A \times A$ – the independence relation.

  In the following, we assume a fixed concurrent alphabet (A, I).
- Define $A^\infty = A^* \cup A^\omega$, i.e., A^∞ is the set of all finite and infinite sequences of elements from A. Define concatenation $\circ$ of elements $u \in A^\infty$ and $v \in A^\infty$ as

$$u \circ v = \begin{cases} u & \text{if } |u| = \omega, \\ uv & \text{else.} \end{cases}$$

  For notational convenience we will write uv instead of $u \circ v$.
- Let $\leqslant_{\text{pref}}$ be the usual prefix ordering on sequences and $\pi_{(a,b)}$ the projection on $\{a, b\}^\infty$. Define a preorder $\preccurlyeq$ on A^∞ which requires the relative order of elements

a and b which are in conflict – i.e., $(a,b) \notin I$ – to be the same when ignoring all other elements of the sequences. Formally,

$u \preccurlyeq v$ if and only if $(\forall (a,b) \notin I.\, \pi_{(a,b)}(u) \leqslant_{\text{pref}} \pi_{(a,b)}(v))$

- Define an equivalence relation $\equiv$ on A^∞ by $u \equiv v$ if and only if $u \preccurlyeq v$ and $v \preccurlyeq u$. The elements of $A^\infty/\equiv$ are called *traces*. The equivalence class of u – the trace containing u – is denoted $[u]$.
- Fact: $\equiv$ is a congruence with respect to $\circ$.
- For $[u],[v] \in A^\infty/\equiv$ define $[u] \preccurlyeq [v]$ if and only if $u \preccurlyeq v$. It can be shown that $\preccurlyeq$ is a partial order over traces. We write $[u] \prec [v]$ if and only if $u \preccurlyeq v$ and $u \not\equiv v$.
- Fact: for $u, v \in A^*$:
  - $[u] \preccurlyeq [v]$ if and only if $(\exists u' \in A^*.\, [uu'] = [v])$
  - $u \equiv v$ if and only if $u \equiv_M v$, where $\equiv_M$ is the well-known "Mazurkiewicz trace equivalence" on finite sequences.

Example 2. Consider the concurrent alphabet (A,I), where $A = \{a,b,c\}$ and $I = \{(a,b)(b,a)\}$. Then, $abc \equiv bac$, $abc \not\equiv acb$, $(abbac)^\omega \equiv (aabbc)^\omega$, and $(abbac)^\omega \not\equiv (abcba)^\omega$.

Remark 3. We have chosen to present traces using projections $\pi_{(a,b)}$ because finite as well as infinite traces are handled in a uniform way. Similar definitions can be found in, e.g., [15, 9].

2.2. Labelled 1-safe nets

We continue by defining *labelled* 1-*safe nets*, the labelled version of 1-safe nets.

Definition 4 (1-*safe nets*). A 1-*safe net*, or just a *net*, is a structure $N = (P,T,F,M_0)$ such that

- P and T are nonempty disjoint countable sets; their elements are called *places* and *transitions*, respectively.
- $F \subseteq (P \times T) \cup (T \times P)$; F is called the *flow relation*.
- $M_0 \subseteq P$; M_0 is called the *initial marking* of N; in general, a set $M \subseteq P$ is called a *marking* or a *state* of N.

Given $a \in P \cup T$, the *preset* of a, denoted ${}^\bullet a$, is defined as $\{a' \mid a'Fa\}$; the *postset* of a, denoted $a^\bullet$, is defined as $\{a' \mid aFa'\}$. The union of ${}^\bullet a$ and $a^\bullet$ is denoted ${}^\bullet a^\bullet$. The irreflexive symmetric *independence relation* I over T is defined by $t_1 I t_2$ if and only if ${}^\bullet t_1^\bullet \cap {}^\bullet t_2^\bullet = \emptyset$. Two transitions t_1 and t_2 are said to be *independent* if $t_1 I t_2$ and in conflict otherwise. Notice that (T,I) is a concurrent alphabet. For $T' \subseteq T$ and $t \in T$ we define $tIT' = T'It = \{t' \in T' \mid t' I t\}$.

Next, we give the definition of firing sequences.

Definition 5 (*Firing sequences*). Let $N = (P,T,F,M_0)$ be a net.

- A transition $t \in T$ is *enabled* at a marking M of N if ${}^\bullet t \subseteq M$ and $t^\bullet \cap (M - {}^\bullet t) = \emptyset$. Denote the set of transitions enabled at a marking M by $next(M)$.
- Given a transition t, define a relation $\xrightarrow{t}$ between markings as follows: $M \xrightarrow{t} M'$ if and only if t is enabled at M and $M' = (M - {}^\bullet t) \cup t^\bullet$. The transition t is said to *occur* (or *fire*) at M. If $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} M_n$ for some markings $M_1, M_2, \ldots, M_n$, then the sequence $\sigma = t_1 \ldots t_n$ is called an *occurrence sequence*. M_n is the marking *reached* by σ, and this is denoted $M_0 \xrightarrow{\sigma} M_n$. A marking M is *reachable* if it is the marking reached by some occurrence sequence. $M \not\rightarrow$ denotes that there are no enabled transitions at M, i.e., $next(M) = \emptyset$, in which case it is said to be *dead* or be a *deadlock*.
- Given a marking M of N, the set of reachable markings of (P, T, F, M) – the net obtained by replacing the initial marking M_0 by M – is denoted by $[M\rangle$.
- A *labelled* 1-safe net $N = (P, T, F, M_0, l)$ is a 1-safe net extended with a labelling function $l : T \to \text{Act}$ mapping each transition to an action in some finite labelling set Act.

The behaviour of a net is captured by its reachability graph.

Definition 6 (*Reachability graph*). The *reachability graph* of a net N is the edge-labelled graph $(V, E)_N$, whose set of vertices (or states), V, is $[M_0\rangle$. The labelled edges are induced by the firing relations $\xrightarrow{t}$, and hence conveniently denoted $M \xrightarrow{t} M'$.

2.3. Partial order semantics

In the following, we assume a fixed labelled 1-safe net N and consider its reachability graph $(V, E)_N$. We use the symbols $p, q, \ldots$ to denote states in $(V, E)_N$. If nothing else is mentioned, it is implicitly assumed that (T, I) is used to generate the congruence $\equiv$.

Definition 7 (*Paths*).

- Define a *path* from $p_0 \in V$ as a sequence, finite or infinite, of transitions $t_1, t_2, \ldots$, for which there exist states $p_1, p_2, \ldots$, such that $p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots$. Notice that the firing rules of the net ensure the uniqueness of the p_i's, if they exist. We therefore refer to $p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots$ as a path from p_0 (also denoted $p_0 \xrightarrow{\sigma}$, where $\sigma = t_1 t_2 \cdots$) and, with a slight abuse of notation, define $path(p_0) \subseteq T^\infty$ to be all paths from p_0.
- Define $comp(p)$ as the maximal elements of $path(p)/\equiv$ with respect to $\preccurlyeq$. For $\sigma \in [\sigma'] \in comp(p)$ we refer to $p \xrightarrow{\sigma}$ as a *computation* from p.

Notice $path(p)$ is limit closed, i.e., if $\sigma_1\gamma_1, \sigma_1\sigma_2\gamma_2, \sigma_1\sigma_2\sigma_3\gamma_3, \ldots \in path(p)$, where all σ_i's are finite, then $\sigma_1\sigma_2 \cdots \in path(p)$. Also, because T is at most countable, $comp(p)$ is always well-defined.

Due to the firing rules of nets, the congruence $\equiv$ respects the property of being a path

Lemma 8. *Given a net* $N=(P,T,F,M_0)$, *and a state* p *of* $(V,E)_N$. *Then,*

$$(\forall \sigma \in path(p).\ (\forall \sigma' \in [\sigma].\ p \xrightarrow{\sigma'}))\ .$$

Proof. If σ is finite, the result easily follows from the commutativity of consecutive independent transitions. If σ is infinite, notice that by interchanging a finite number of consecutive independent transitions of σ we conclude that any finite prefix of σ' is an element of $path(p)$. Since $path(p)$ is limit closed, we conclude $\sigma' \in path(p)$. □

Hence, $path(p)$ can be partitioned into elements of $T^\infty/\equiv$. Moreover, if σ is finite, then $p \xrightarrow{\sigma} q$ implies $(\forall \sigma' \in [\sigma].\ p \xrightarrow{\sigma'} q)$.

Definition 9 (*Continuously concurrently enabled transitions*). Given $\sigma \in path(p_0), |\sigma| = \omega, \sigma = t_1 t_2 \cdots$. A transition t is said to be *continuously concurrently enabled* (cc-enabled) along $p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots$ if and only if t is enabled at some p_i, $i \geqslant 0$, and independent of the remaining transitions of σ from that state. Formally, $(\exists n \in \mathbb{N}.\ (\forall j \geqslant n.\ p_j \xrightarrow{t} \wedge\, t I t_{j+1}))$. Notice that the irreflexivity of I implies that $\forall n,\ t \neq t_j$, $j \geqslant n$. Whenever p_0 is clear from the context, t is said to be cc-enabled along σ.

Example 10. In the process agent example from Fig. 1, c is cc-enabled along $i \xrightarrow{a^\omega}$, when we use a, b, and c to refer to the corresponding transitions. Also, bca^ω is a computation from i, while a^ω is not.

The next two lemmas state properties of traces.

Lemma 11. *Given a concurrent alphabet* (A,I). *For* $\sigma, \sigma' \in A^\infty$ *we have that*

$$\sigma \preccurlyeq \sigma' \Leftrightarrow (\forall \sigma_1 \in pref_{\text{fin}}(\sigma).\ (\exists \sigma_1' \in pref_{\text{fin}}(\sigma').\sigma_1 \preccurlyeq \sigma_1'))\ ,$$

where $pref_{\text{fin}}(\sigma)$ *denotes the finite prefixes of* σ.

Proof. The "if" direction is proved by an easy contradiction argument. For the "only if" direction, first choose a finite prefix σ_1' of σ' such that its Parikh vector (for each $a \in A$ this vector provides the number of occurrences of a's in σ_1') is greater than or equal to that of σ_1. Assuming $\sigma_1 = a_1 \cdots a_n$ and $\sigma_1' = b_1 \cdots b_m$ find the first occurrence of a_1, say b_{j_1} in σ_1'. Then for any $1 \leqslant j < j_1$ it must be the case that $b_j I b_{j_1}$, since we have $\sigma \preccurlyeq \sigma'$. Hence, $b_1 \cdots b_m \equiv b_{j_1} b_1 \cdots b_{j_1-1} b_{j_1+1} \cdots b_m$. Continuing this procedure for $a_2, \ldots, a_n$ we eventually get that $\sigma_1' \equiv \sigma_1 \gamma$ for some $\gamma \in A^*$. But then clearly $\sigma_1 \preccurlyeq \sigma_1'$. □

Lemma 12. *Given a net $N=(P,T,F,M_0)$, a state p of $(V,E)_N$, $\sigma\in path(p)$ such that $|\sigma|=\omega$, and $t\in T$ that is cc-enabled along σ. Then, for any $\sigma'\in[\sigma]$, t is cc-enabled along σ', i.e., $\equiv$ respects cc-enabledness.*

Proof. Clearly, by definition there exists a finite $\sigma_1\in path(p)$, a $p'\in S$, and a $\sigma_2\in path(p')$ such that $p\xrightarrow{\sigma}=p\xrightarrow{\sigma_1}p'\xrightarrow{\sigma_2}$ and t is enabled at p' and independent of all transitions in σ_2. Choose any $\sigma'\in[\sigma]$. Since $\sigma\preccurlyeq\sigma'$, it follows from Lemma 11 that there exists a finite prefix of σ', say σ_1', such that $\sigma_1\preccurlyeq\sigma_1'$. Using the technique from the proof of Lemma 11 we see that there exists a $\gamma\in T^*$, such that $\sigma_1\gamma\equiv\sigma_1'$ and all transitions in γ are independent of t. We conclude that t must be enabled at p'', where $p\xrightarrow{\sigma_1'}p''$, since $p\xrightarrow{\sigma_1\gamma}p''$. Choosing σ_2' such that $\sigma'=\sigma_1'\sigma_2'$ we also conclude that all transitions in σ_2' are independent of t, since all transitions in σ_2' occur in σ_2. Hence, t is cc-enabled along σ'. □

Hence, based on Lemma 12 we may safely write $t\in T$ is cc-enabled along $[\sigma]$, meaning t is cc-enabled along $\sigma\in path(p)$.

Next, we identify maximal traces as maximal elements in a partial order. The following lemma explains why we focus on these traces. They can be thought of representing executions of a concurrent system which are fair with respect to progress of independent processes. In [21] the term "concurrency fairness" is used for such behaviours. Compared to other notions of "fairness" in the context of concurrent systems "progress fairness" is a weak assumption, see [18] for a comparison to other notions of fairness.

Lemma 13. *Given a net $N=(P,T,F,M_0)$ and a state p of $(V,E)_N$. For $[\sigma]\in comp(p)$ such that $|\sigma|=\omega$ we have*

$$(\exists[\sigma']\in comp(p).[\sigma]\prec[\sigma'])\Leftrightarrow(\exists t\in T.\ t\ \textit{is cc-enabled along}\ \sigma)\ .$$

Proof. The "if" direction is easy, and hence omitted. For the "only if" direction first observe the following: since $[\sigma]\prec[\sigma']$, there must exist a $t\in T$ such that $\pi_{(t,t)}(\sigma)<\pi_{(t,t)}(\sigma')$. Clearly, $|\pi_{(t,t)}(\sigma)|=n<\omega$ for some $n\in\mathbb{N}$. Let $\sigma=\sigma_1\sigma_2$, where $\#_t(\sigma_1)=n,\#_t(\sigma_2)=0$, $|\sigma_1|<\omega$, and $\#_t(\sigma)$ is the number of t's in σ. By Lemma 11 we know that there exists a finite prefix σ_3 of σ' such that $[\sigma_1]\preccurlyeq[\sigma_3]$. Furthermore, there must exists a suffix of σ such that all transitions of it are independent of t. To see this, assume that there were infinitely many indexes $i_j\in\mathbb{N}$ for $0\leqslant j$ such that $(t_{i_j},t)\notin I$, where $\sigma=t_1t_2\cdots$. Since $(\forall j\in\mathbb{N}.\ \pi_{(t,t_{i_j})}(\sigma)<_{\text{pref}}\pi_{(t,t_{i_j})}(\sigma'))$, all t_{i_j}'s must occur before the $(n+1)$th t in $\pi_{(t,t_{i_j})}(\sigma')$. But this clearly means that there must be infinitely many transitions between the nth and $(n+1)$th t in σ', which is impossible.

Next, we show that there must exist a transition t' which is cc-enabled along σ. First, choose the first occurrence of a $t'\in T$ along σ' such that $\#_{t'}(\sigma)<\#_{t'}(\sigma')$. Next, split σ into σ_1 (finite) and σ_2 such that $\sigma=\sigma_1\sigma_2$ and all transitions in σ_2 are independent of t'. Then, choose σ_1' as the shortest prefix of σ' such that $\#_{t'}(\sigma_1')\geqslant\#_{t'}(\sigma_1)+1$ and the

Parikh vector of σ_1' is greater than that of σ_1. By an argument similar to that above, one can rearrange σ_1' by continuously interchanging adjacent independent transitions and obtain $\sigma_1' \equiv \sigma_1\gamma \in path(p)$. Now $\#_{t'}(\gamma) > 0$. Let $\gamma = t_1' \cdots t_r' t' t_1'' \cdots t_s''$, where $r,s \geqslant 0$ and all t_i''s are different from t'. Now assume that $(\exists 1 \leqslant j \leqslant r.(t_j', t') \notin I)$. Choose the first such j. Then, $\pi_{(t_j',t')}(\sigma_1') >_{\text{pref}} \pi_{(t_j',t')}(\sigma_1)$ and since the relative occurrence of t' and t_j''s in σ_1' and $\sigma_1\gamma$ are the same, a t_j' must occur before the $(\#_{t'}(\sigma_1)+1)$'th t' in σ'. But $\#_{t_j'}(\sigma) = \#_{t_j'}(\sigma')$ by choice of t'. Then, there must exist a t_j' in σ_2 and this contradicts the assumption that all transitions in σ_2 were independent of t'. By using the properties of $(V,E)_N$ and I (e.g., permutation of consecutive independent transitions: if $M \xrightarrow{t} M' \xrightarrow{t'} M''$ and tIt', then there exists an M''' such that $M \xrightarrow{t'} M''' \xrightarrow{t} M''$),[3] we conclude that t' must be enabled at p' where $p \xrightarrow{\sigma_1} p'$. Hence, t' is cc-enabled along σ. □

3. The logic P-CTL and its interpretation

In this section, we assume a fixed labelled 1-safe net $N = (P, T, F, M_0, l)$. The syntax of the logic P-CTLis

$$A ::= tt \mid \neg A \mid A_1 \wedge A_2 \mid \bigcirc_\alpha A \mid A_1 \, U_\exists \, A_2 \mid A_1 \, U_\forall \, A_2,$$

where $\alpha \in Act$ and tt is an abbreviation for *true*.

In Hennessy–Milner logic [22], $\langle\alpha\rangle A$ expresses the fact that one can perform an action α from a state and, in doing so, reach another state at which A holds. Here, the $\bigcirc_\alpha A$ expresses that a transition labelled α can be fired reaching a state where A holds. The logic is interpreted over the reachability graph $(V,E)_N$ of N as follows, where $p \in V$, $\alpha \in Act$, and we have written $\models$ instead of $\models_N$.

- $p \models tt$.
- $p \models \neg A$ if and only if $p \not\models A$.
- $p \models A_1 \wedge A_2$ if and only if $p \models A_1$ and $p \models A_2$.
- $p \models \bigcirc_\alpha A$ if and only if $(\exists t \in T, q \in V.\ l(t) = \alpha \wedge p \xrightarrow{t} q \wedge q \models A)$.
- $p \models A_1 \, U_\exists \, A_2$ if and only if $(\exists [\sigma] \in comp(p),\ p \xrightarrow{\sigma} = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots (\exists 0 \leqslant n \leqslant |\sigma|.(p_n \models A_2) \wedge (\forall 0 \leqslant i < n.\ p_i \models A_1)))$.
- $p \models A_1 \, U_\forall \, A_2$ if and only if $(\forall [\sigma'] \in comp(p).\ (\forall \sigma \in [\sigma'],\ p \xrightarrow{\sigma} = p \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots (\exists 0 \leqslant n \leqslant |\sigma|.(p_n \models A_2) \wedge (\forall 0 \leqslant i < n.\ p_i \models A_1))))$.

Furthermore, we define $ff \equiv \neg tt$, $\langle\alpha\rangle A \equiv \bigcirc_\alpha A$, $[\alpha]A \equiv \neg\langle\alpha\rangle\neg A$, $\mathsf{F}(A) \equiv tt \, U_\exists \, A$, $\mathsf{G}(A) \equiv \neg\mathsf{F}(\neg A)$, $\mathsf{Ev}(A) \equiv tt \, U_\forall \, A$, and $\mathsf{Al}(A) \equiv \neg\mathsf{Ev}(\neg A)$. The intended meaning of $\mathsf{Ev}(A)$ is that eventually/inevitably A will hold along any computation, while $\mathsf{Al}(A)$ means that along some computation A always holds.

[3] To be more precise, we use the axioms of the corresponding labelled asynchronous transition system, which intuitively is $(V,E)_N$ augmented with I [39].

Notice that the "until" operators $U_\exists$ and $U_\forall$ quantify over computations rather than paths and that path quantified formulas are not necessarily interpreted over a limit closed set of paths as it is the case with the standard interpretation of CTL.

Example 14. In the process agent example from Fig. 1 we have $i \models \mathsf{Ev}(\langle c \rangle tt)$.

Having given the necessary definitions, we end this section by defining the model-checking problem.

Definition 15. Given a labelled 1-safe net $N = (P, T, F, M_0, l)$ and a formula A. The *model-checking problem of N and A* is the problem of deciding whether or not $M_0 \models A$ in $(V, E)_N$.

4. Model-checking by state labelling

In this section we present a state labelling based algorithm that solves the model-checking problem. The algorithm essentially works as the one presented for CTL in [8] except for the $U_\forall$ operator.

Theorem 16. *Given a net N and a formula A. Let $(V, E)_N$ denote the reachability graph of $N = (P, T, F, M_0, l)$. The following state labelling based algorithm solves the model-checking problem for N and A in time* $O(|A|(|V| + |E||T|))$.

Proof. Given a formula A and a net N, the algorithm proceeds in stages as follows. In the first stage all subformulas of length one are processed. In general, at stage i all subformulas of length i are processed and at the end of stage i a state is labelled with a subformula A' of length i of A (or its negation $\neg A'$) if and only if it is satisfied in that state. Hence, after the $|A|$th stage all states in V will have been labelled with either A or $\neg A$.

The data structures needed to perform the labelling are essentially those described for the CTL model-checker in [8]. The only exception is the $U_\forall$ operator ($U_\exists$ can be handled as the EU operator in CTL, since any finite prefix of a path can be extended to a computation.). The $U_\forall$ operator is handled as follows.

Assume we want to label the states with the subformula $A' = A_1 U_\forall A_2$. All states must already have been labelled appropriately with A_1, $\neg A_1$, A_2, and $\neg A_2$. Then, states labelled with A_2 are labelled with A', and states labelled with $\neg A_1$ and $\neg A_2$ are labelled with $\neg A'$. The remaining states must then all be labelled with A_1 and $\neg A_2$.

Compute the maximal strongly connected components of (V, E) restricted to these remaining states. Let us denote the graph whose nodes are these maximal strongly connected components by G'. G' is a directed acyclic graph (DAG) whose nodes are sets of states of V. Now, as long as there is a terminal node n in G', repeatedly do the following.

(1) If there is a state $p \in n$, a transition $t \in T$, and a state $p' \in V$ such that $p \xrightarrow{t} p'$ and p' is labelled with $\neg A'$, then label all states in n with $\neg A'$. Furthermore, for all nodes m in G', if n can be reached from m then label all states in m with $\neg A'$. Remove all processed nodes (i.e., nodes newly labelled $\neg A'$) from G' and let G' denote the new DAG.

(2) Else, if there is a state p in n but no transition t and state $p' \notin n$ such that $p \xrightarrow{t} p'$, then label all states in n (and m's above n, as described just above) with $\neg A'$ (there must exist an invalidating computation in n from p). Update G' as above.

(3) Else, all states of n have successor states not in n. Moreover, these successor states are all labelled by A'. Assume $T = \{t_1, \ldots, t_k\}$.

- Initialise a boolean array B of length k such that all its entries are set to False. Then, for each edge $\xrightarrow{t_i}$ between any two states in n, set all entries $B[j]$ such that $\neg(t_i I t_j)$ to True.
- If there is an entry $B[l]$ which is False and t_l is enabled at any state in n, then label all states in n with A'. Remove n from G' and let G' denote the new DAG.
- Else, label all states in n (and m's, as described in the first case) with $\neg A'$ and update G' as above.

It should be obvious that case (1) labels the states in n correctly. Case (2) is also correct because we can exhibit a computation in n whose states are labelled with A_1 and $\neg A_2$. Case (3) is correct because of the following observation: there exists a computation inside n if and only if there is no transition t_l that is (i) independent of all transition labelling edges between states in n, and (ii) enabled at (necessarily all) a state in n.

An analysis of the algorithm yields the time complexity $O(|A|(|V| + |E||T|))$. Hence, our algorithm is comparable to the one presented in [8]. □

5. A tableau method for model-checking

In this section we present a local model-checker based on a tableau system for model-checking formulas from our logic.

Local model-checking based on tableau systems has been presented in, e.g., [16, 33]. As opposed to a global model-checker – as the one presented in the previous section – which checks if all states of the system satisfy a formula, a local model-checker only checks if a specific state satisfies a given formula. For local model-checkers based on tableau systems this is done by only visiting (other) states if the tableau rules require it. Hence, the local model-checker may well be able to show that a state satisfies a formula without visiting all states of the system. For systems such as 1-safe nets a local model-checker can thus postpone the generation of the entire reachability graph and only generate the parts the tableau rules require. Since the size of the reachability graph can be exponentially bigger than the size of the net, a local model-checker sometimes has an advantage over a global model-checker, since it might avoid the so-called "state-space explosion problem".

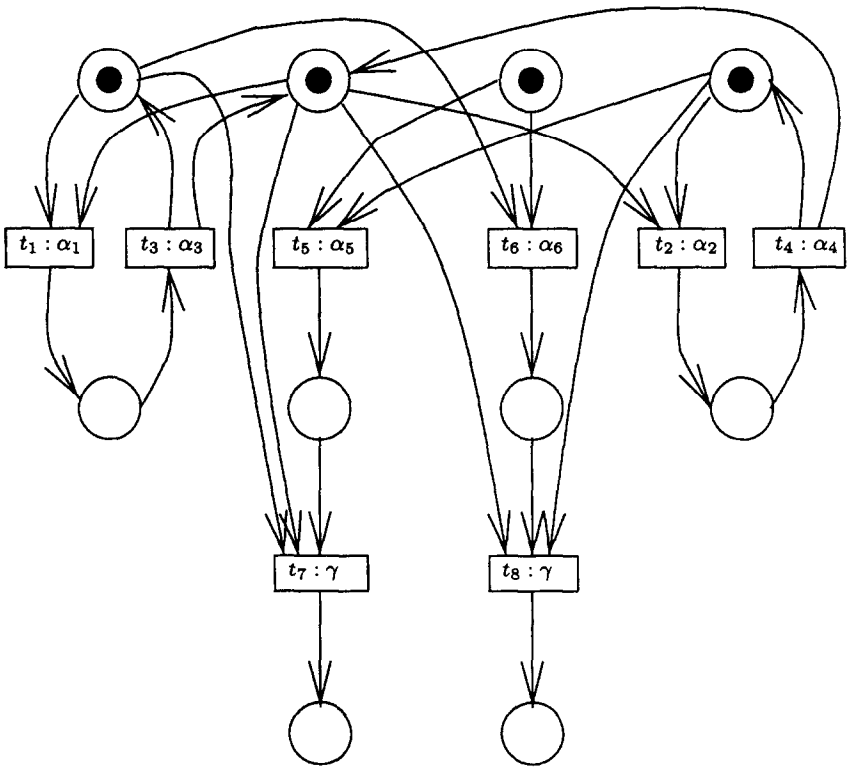
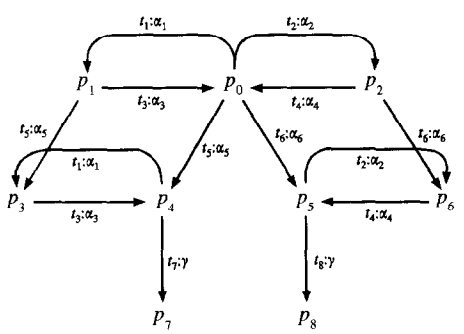


Fig. 3. The net N_1.

Below we consider a very simple reachability graph g_1, which is generated by the 1-safe net N_1 in Fig. 3.



The t_i's are the transitions, the Greek letters the labels, and p_0 the initial marking. The independence relation is the smallest such containing $(t_1, t_5), (t_3, t_5), (t_2, t_6)$, and (t_4, t_6). Clearly, $p_0 \models_{N_1} \neg \mathsf{Ev}(\langle\gamma\rangle tt)$ since $[(t_1 t_3 t_2 t_4)^\omega] \in comp(p_0)$ and no state along the computation $(t_1 t_3 t_2 t_4)^\omega$ satisfies $\langle\gamma\rangle tt$. However, if we drop the transitions t_2, t_4, t_6, and t_8 and call this reduced net N_2, we do indeed have $p_0 \models_{N_2} \mathsf{Ev}(\langle\gamma\rangle tt)$, since every

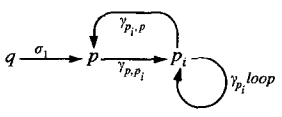
computation from p_0 must eventually reach $p_4 - t_5$ cannot be continuously ignored while repeatedly firing t_1 and t_3 since they are both *independent* of t_5.

5.1. Tableau rules

In the following we consider a fixed labelled 1-safe net N and its reachability graph $(V,E)_N$.

We want to perform local model-checking by unfolding parts of the reachability graph into a tree structure. The tableau rules are supposed to guide this unfolding by imposing constraints which restrict the size and shape of the tree structure. The main difficulty is handling the $U_\forall$ operator.

Consider a state q such that $q \not\models A_1 U_\forall A_2$. Then either there exists (i) a computation σ such that $A_1 \wedge \neg A_2$ holds at all states along $q \xrightarrow{\sigma}$ until either a deadlock is reached or a state such that $\neg A_1 \wedge \neg A_2$ holds reached, or (ii) an infinite computation σ such that $A_1 \wedge \neg A_2$ holds at all states along $q \xrightarrow{\sigma}$, referred to as an invalidating computation. Since the formulas are interpreted at states and the state space is finite, case (ii) reduces (simply by removing a finite number of loops from σ) to the existence of an infinite computation $\sigma_1\sigma_2$ from q, where σ_1 is finite and all states along $q \xrightarrow{\sigma_1}$ occur only once along $q \xrightarrow{\sigma_1} p \xrightarrow{\sigma_2}$, while all states along $p \xrightarrow{\sigma_2}$ occur infinitely often. Notice $A_1 \wedge \neg A_2$ still holds at all states, as will be the case for the following computations. Using Lemma 13 it is possible to obtain from σ_2 an infinite path σ_3 from p of the form $(\gamma_{p,p_1}\gamma_{p_1 loop}\gamma_{p_1,p}\cdots\gamma_{p,p_k}\gamma_{p_k loop}\gamma_{p_k,p})^\omega$, where all γ's are finite and made up from subsequences of σ_2 and $1 \leqslant k \leqslant |T|$. The indices are intended to illustrate the structure of the loops as follows:



Since σ_2 was a computation from p, the γ's can be chosen such that for any $t \in next(p)$ one of the $\gamma_{p_i loop}$'s will contain a transition in conflict with t. Hence, σ_3 is a computation from p. We refer to the illustrated loops $\gamma_{p,p_i}\gamma_{p_i loop}\gamma_{p_i,p}$ as *critical loops*. To conclude, $\sigma_1\sigma_3$ is an invalidating computation from q along which all states satisfy $A_1 \wedge \neg A_2$.

In the example from Section 5, if we chose p_0 as p, then $p_0 \xrightarrow{t_1}\xrightarrow{t_3} p_0$ and $p_0 \xrightarrow{t_2}\xrightarrow{t_4} p_0$ would constitute critical loops. Actually, the sizes of the γ's can be bound since the state space is finite. The important observation is that together with $|T|$ we obtain a bound on the length and number of γ's we have to consider. The bounds will be encoded in an annotated logic.

5.1.1. The annotated logic

The syntax of the annotated logic used in the tableau rules differs from that of P-CTL only with respect to the $U_\exists$ and $U_\forall$ operators, which are replaced by labelled

counterparts. The $U_\exists$ operator is replaced by $U_\exists^{\mathsf{C}}$, where $\mathsf{C} \subseteq V$. The intuition is that C keeps track of which states have been visited and prevents unnecessary unfolding. For the $U_\forall$ operator we use a more elaborate annotation, $U_\forall^{\mathsf{C}}$, $U_\forall^{(p,n,T')}$, $U_\forall^{(p,n,T',V',\rightarrow)}$, and $U_\forall^{(p,n,T',V',\leftarrow)}$, where $p \in V$, $T' \subseteq T$, $V' \subseteq V$, and $0 \leqslant n \leqslant |T|$. V' plays a role similar to C, n bounds the number of critical loops the tableau rules allow to explore, and T' keeps track of which transitions have been concurrently enabled but ignored so far along a path. The emptiness of T' will indicate that an invalidating computation has been found.

Let *Ann* be the obvious homomorphism which annotates a formula A (generated by the grammar in Section 3) by transforming every $U_\exists$ and $U_\forall$ into $U_\exists^{\emptyset}$ and $U_\forall^{\emptyset}$, respectively. An annotated formula B is said to be *clean* if there exists a formula A such that B equals $Ann(A)$.

5.1.2. *The tableau rules*

The tableau rules will consist of rules for sequents of the form $p \vdash B$. The rules can be read from top to bottom as: "the top sequent (or conclusion) holds (B holds at p) if the bottom sequents (or antecedents) and side conditions hold". B, B_1, and B_2 are assumed to be clean annotated formulas.

(1) $$\frac{p \vdash B_1 \wedge B_2}{p \vdash B_1 \quad p \vdash B_2}$$

(2) $$\frac{p \vdash \bigcirc_\alpha B}{q \vdash B}$$

where $t \in T, q \in V, p \xrightarrow{t} q, l(t) = \alpha$

(3) $$\frac{p \vdash B_1 \, U_\exists^{\mathsf{C}} \, B_2}{p \vdash B_2}$$

where $p \notin \mathsf{C}$

(4) $$\frac{p \vdash B_1 \, U_\exists^{\mathsf{C}} \, B_2}{p \vdash B_1 \quad q \vdash B_1 \, U_\exists^{\mathsf{C} \cup \{p\}} \, B_2}$$

where $p \notin \mathsf{C}, t \in T, q \in V, p \xrightarrow{t} q$

(5) $$\frac{p \vdash B_1 \, U_\forall^{\mathsf{C}} \, B_2}{p \vdash B_2}$$

where $p \notin \mathsf{C}$

(6) $$\frac{p \vdash B_1 \, U_\forall^{\mathsf{C}} \, B_2}{p \vdash B_1 \quad q_1 \vdash B_1 \, U_\forall^{\mathsf{C} \cup \{p\}} \, B_2 \cdots q_m \vdash B_1 \, U_\forall^{\mathsf{C} \cup \{p\}} \, B_2}$$

where $next(p) = \{t_1, \ldots, t_m\}$, $0 < m \in \mathbb{N}$, $p \notin \mathsf{C}$, $(\forall\, 1 \leqslant i \leqslant m.\ p \xrightarrow{t_i} q_i)$

(7) $$\frac{p \vdash B_1 \, U_\forall^{\mathsf{C}} \, B_2}{p \vdash B_1 \, U_\forall^{(p, |next(p)|, next(p))} \, B_2}$$

where $p \in \mathbf{C}$

$$(8) \qquad \frac{p \vdash B_1\, U_\forall^{(p,n,T')}\, B_2}{p \vdash B_1\, U_\forall^{(p,n-1,T',\emptyset,\rightarrow)}\, B_2}$$

where $0 < n \in \mathbb{N}, T' \neq \emptyset$

$$(9) \qquad \frac{q \vdash B_1\, U_\forall^{(p,n,T',V',\rightarrow)}\, B_2}{q \vdash B_1 \;\; q_i \vdash B_1\, U_\forall^{(p,n,t_i IT',V' \cup \{q\},\rightarrow)}\, B_2}$$

where $q \notin V', next(q) = \{t_1, \ldots, t_m\}$, $0 < m \in \mathbb{N}, (\forall\, 1 \leqslant i \leqslant m . q \xrightarrow{t_i} q_i)$

$$(10) \qquad \frac{q \vdash B_1\, U_\forall^{(p,n,T',V',\rightarrow)}\, B_2}{q \vdash B_2}$$

where $q \notin V'$

$$(11) \qquad \frac{q \vdash B_1\, U_\forall^{(p,n,T',V',\rightarrow)}\, B_2}{q \vdash B_1\, U_\forall^{(p,n,T',\emptyset,\leftarrow)}\, B_2}$$

where $q \in V'$

$$(12) \qquad \frac{q \vdash B_1\, U_\forall^{(p,n,T',V',\leftarrow)}\, B_2}{q \vdash B_1 \;\; q_i \vdash B_1\, U_\forall^{(p,n,t_i IT',V' \cup \{q\},\leftarrow)}\, B_2}$$

where $q \notin V', next(q) = \{t_1, \ldots, t_m\}$, $0 < m \in \mathbb{N}, q \neq p, (\forall\, 1 \leqslant i \leqslant m . q \xrightarrow{t_i} q_i)$

$$(13) \qquad \frac{q \vdash B_1\, U_\forall^{(p,n,T',V',\leftarrow)}\, B_2}{q \vdash B_2}$$

where $q \notin V'$

$$(14) \qquad \frac{p \vdash B_1\, U_\forall^{(p,n,T',V',\leftarrow)}\, B_2}{p \vdash B_1\, U_\forall^{(p,n,T')}\, B_2}$$

Rules 1–4 need no further explanation. Referring to the notation from Section 5.1, Rules 5 and 6 should detect σ_1, Rule 7 should detect the "switch" to σ_3, Rules 8–10 should detect $\gamma_{p,p_i}\gamma_{p_i loop}$, Rule 11 should detect the "switch" to $\gamma_{p_i,p}$, and Rules 12–14 should detect $\gamma_{p_i,p}$.

The next step is to define derivation trees which are built up according to the tableau rules.

5.1.3. The derivation trees and tableaux

In this section we define the tableaux. This is done by first defining a larger class of trees, derivation trees, which are generated according to the tableau rules. The next step is to restrict the class of derivation trees, using the annotation of the formulas, to a subclass of derivation trees which will be defined to be the tableaux.

Derivation trees are defined inductively in the usual manner, except perhaps for case of negated formulas. That is, if $\mathsf{T}_1, \ldots, \mathsf{T}_n$ are derivation trees with roots matching

the antecedents of a rule and the side conditions are fulfilled, then one obtains a new derivation tree by "pasting the derivation trees together" according to the rule. The root of the new derivation tree is labelled by the conclusion of the rule. A tree consisting of a single node labelled with one of the following sequents is a derivation tree:

- $p \vdash tt$.
- $p \vdash \neg B$.
- $p \vdash B_1\, U_\forall^{(p,n,T')}\, B_2$, where $n=0$ or $T'=\emptyset$.
- $q \vdash B_1\, U_\forall^{(p,n,T',V',\leftarrow)}\, B_2$, where $q \in V'$.

By applying the rules we can obtain new derivation trees, for example:

- If T_1 is a derivation tree with root $p \vdash B_1$, T_2 is a derivation tree with root $q \vdash B_1\, U_\exists^{\mathsf{C}\cup\{p\}}\, B_2$, where $p \notin \mathsf{C}$, and there exists a $t \in T$ such that $p \xrightarrow{t} q$, then

$$\frac{p \vdash B_1\, U_\exists^{\mathsf{C}}\, B_2}{\mathsf{T}_1\,\mathsf{T}_2}$$

  is a derivation tree with root $p \vdash B_1 U_\exists^{\mathsf{C}} B_2$.
- If T is a derivation tree with root $p \vdash B_2$ and $p \notin \mathsf{C}$, then

$$\frac{p \vdash B_1\, U_\forall^{\mathsf{C}}\, B_2}{\mathsf{T}}$$

  is a derivation tree with root $p \vdash B_1\, U_\forall^{\mathsf{C}}\, B_2$.

Nothing else is a derivation tree.

We continue by defining the tableaux. In this step we get rid of derivation trees as for example $p \vdash \neg tt$. Sequents of the form $q \vdash B_1\, U_\forall^{(q,n,\emptyset)}\, B_2$, where $n \in \mathbb{N}$ and $q \in V$, are called terminal sequents. A tableau is a derivation tree T with root $p \vdash Ann(A)$ such that either

- $A = tt$ or
- $A = \neg A'$ and there exists no tableau with root $p \vdash Ann(A')$ or
- A is not of the above form and (a) every proper subtree T' of T whose root is labelled with a clean formula is itself a tableau and (b) T has no leaves labelled with terminal sequents.

A sequent $p \vdash B$ is *proved* by exhibiting a tableau with root $p \vdash B$.

5.2. Soundness and completeness

Having given the necessary definitions we are now ready to state the main result.

Theorem 17. *Given a finite labelled net $N=(P,T,F,M_0,l)$, then for any state p of $(V,E)_N$ $(p \in V)$ and any formula A we have*

$$p \models A \text{ if and only if there exists a tableau with root } p \vdash Ann(A)$$

Proof. The proof proceeds by structural induction in A, showing soundness and completeness simultaneously. The main difficulty is the $U_\forall$ operator. For the soundness part, our observations from Section 5.1 provide the basis for a proof by contradiction.

For the completeness part, using the induction hypothesis one can give a direct construction of a tableau. Intuitively, if $p \models A_1 U_\forall A_2$, then a tableau will be constructed (top-down from p) by always proving $q \vdash Ann(A_2)$ if $q \models A_2$ for any reached state q. Else, if $q \not\models A_2$, then one proves $q \vdash Ann(A_1)$, starts unfolding the graph from q, and continues by trying to prove $Ann(A_1 U_\forall A_2)$ at the states that are reached.

Case tt: Clearly, we always have $p \models tt$ and the tableau $p \vdash tt$.

Case $\neg$: We have $p \models \neg A$ if and only if $p \not\models A$ if and only if (induction hypothesis) there exists no tableau with root $p \vdash Ann(A)$ if and only if $p \vdash \neg Ann(A)$ is a tableau.

Case $\wedge$: We have $p \models A_1 \wedge A_2$ if and only if $p \models A_1$ and $p \models A_2$ if and only if (induction hypothesis) there exists tableaux T_1 with root $p \vdash Ann(A_1)$ and T_2 with root $p \vdash Ann(A_2)$ if and only if there exists a tableau with root $p \vdash Ann(A_1 \wedge A_2)$, because $Ann(A_1 \wedge A_2) = Ann(A_1) \wedge Ann(A_2)$.

Case $\bigcirc_\alpha$: We have $p \models \bigcirc_\alpha A$ if and only if $(\exists t \in T, q \in V. p \xrightarrow{t} q \wedge q \models A \wedge l(t) = \alpha)$ if and only if (induction hypothesis) there exists a tableau T and $(\exists t \in T, q \in V. p \xrightarrow{t} q \wedge l(t) = \alpha \wedge \mathsf{T}$ has root $q \vdash Ann(A))$ if and only if there exists a tableau T with root $p \vdash Ann(\bigcirc_\alpha A)$, since $Ann(\bigcirc_\alpha A) = \bigcirc_\alpha Ann(A)$.

Case $U_\exists$: We have $p \models A_1 U_\exists A_2$ if and only if $(\exists p_1, p_2, \ldots, p_n \in V,\ t_1, t_2, \ldots, t_n \in T, n \geqslant 0. p = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots \xrightarrow{t_n} p_n \wedge p_n \models A_2 \wedge (\forall 0 \leqslant i < n. p_i \models A_1) \wedge (\forall 0 \leqslant i < j \leqslant n. p_i \neq p_j))$ if and only if (induction hypothesis) there exists tableaux $\mathsf{T}_0, \ldots, \mathsf{T}_{n-1}$ with roots $p_0 \vdash Ann(A_1), \ldots, p_{n-1} \vdash Ann(A_1)$ and T_n with root $p_n \vdash Ann(A_2)$ and transitions $t_1, \ldots, t_n$ such that $p = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots \xrightarrow{t_n} p_n$ is loop free if and only if there exists a tableau with root $p \vdash Ann(A_1) U_\exists^\emptyset Ann(A_2)$, because $Ann(A_1 U_\exists A_2) = Ann(A_1) U_\exists^\emptyset Ann(A_2)$.

Case $U_\forall$: We show the bi-implication by showing the left and right implications separately.

"*Only if*" *direction* (*completeness*): We show one can obtain a derivation tree with root $p \vdash Ann(A_1 U_\forall A_2)$. This will be done by providing an algorithm which will be shown to terminate and produce the desired tree. We then argue that it is a tableau.

The tree will be constructed from the root and expanded downwards. Only so-called "active" leaves of the current tree will be expanded. We try to keep the tree as small as possible by first trying to prove that B_2 holds at a state. Only if this is not possible do we expand the tree.

During the presentation of the algorithm several claims are made. All of them will be shown to be valid in the succeeding paragraph. For convenience, we write B_1 for $Ann(A_1)$ and B_2 for $Ann(A_2)$. So $B_1 U_\forall^\emptyset B_2 = Ann(A_1 U_\forall A_2)$. The algorithm consists of the following steps:

Step 1: Start by creating the root which is labelled by $p \vdash B_1 U_\forall^\emptyset B_2$. Mark this node as *active*.

Step 2: If possible choose an active node N, labelled by a sequence of one of the following forms:

(i) $q \vdash B_1 U_\forall^C B_2$,

(ii) $q \vdash B_1\ U_\forall^{(q,n,T')}\ B_2$,
(iii) $q \vdash B_1\ U_\forall^{(p,n,T',S',\leftrightarrow)}\ B_2$,
where $\leftrightarrow$ stands for either $\leftarrow$ or $\rightarrow$. Else terminate.

Step 3: If $q \models A_2$, then by induction we have the existence of a tableau T' with root $q \vdash B_2$. Deactivate N and paste T' below N using rules 5, 10, or 13. None of the added nodes are active. Note that $q \models A_2$ excludes (ii) because of the way the current tree has been expanded.

Step 4: Else if $q \models \neg A_2$, then necessarily (Claim 1) $q \models A_1$. By induction there exists a tableau T' with root $q \vdash B_1$.

- If N is of the form (i), and $q \notin \mathsf{C}$, then (Claim 2) $next(q) \neq \emptyset$ and apply rule 6, using T'. Deactivate N and activate the new leaves labelled $q_i \vdash B_1\ U_\forall^{\mathsf{C}\cup\{q\}}\ B_2$ that were added by application of rule 6.
- If N is of the form (i) and $q \in \mathsf{C}$, then deactivate N and, using rule 7, add a node below N labelled $q \vdash B_1\ U_\forall^{(q,|T|,next(q))}\ B_2$. Using rule 8, because (Claim 3) $next(q) \neq \emptyset$, add yet another node below labelled $q \vdash B_1\ U_\forall^{(q,|T|-1,next(q),\emptyset,\rightarrow)}\ B_2$ which is activated.
- If N is of the form (ii), then (Claim 4) $T' \neq \emptyset$. If $n = 0$, then deactivate N. Else if $n > 0$, then deactivate N and apply rule 8, adding a node below N labelled $q \vdash B_1\ U_\forall^{(q,n-1,T',\emptyset,\rightarrow)}$. Activate this node.
- If N is of the form (iii) ($\rightarrow$) and $q \notin S'$, then (Claim 5) $next(q) \neq \emptyset$ and we deactivate N. By induction we have the existence of the tableau T' with root labelled $q \vdash B_1$. Using rule 9 add this tree below N and add nodes labelled $q_i \vdash B_1 U_\forall^{(p,n,t_i IT',S'\cup\{q\},\rightarrow)} B_2$. Only the last nodes are activated.
- If N is of the form (iii) ($\rightarrow$) and $q \in S'$, then deactivate N and using rule 11 add a node below N labelled $q \vdash B_1\ U_\forall^{(p,n,T',\emptyset,\leftarrow)}\ B_2$. Activate this node.
- If N is of the form (iii) ($\leftarrow$), $q \notin S'$, and $q \neq p$, then deactivate N. Because (Claim 6) $next(q) \neq \emptyset$, we can use rule 12 and the induction hypothesis to add a tableau T' with root labelled $q \vdash B_1$. Also, add nodes labelled $q_i \vdash B_1\ U_\forall^{(p,n,t_i IT',S'\cup\{q\},\leftarrow)}\ B_2$. Only these last nodes will be activated.
- If N is of the form (iii) ($\leftarrow$) and $q \in S'$ and $q \neq p$, then deactivate N.
- If N is of the form (iii) ($\leftarrow$) and $q = p$, then apply rule 14. Deactivate N and activate the added node labelled $q \vdash B_1\ U_\forall^{(q,n,T')}\ B_2$

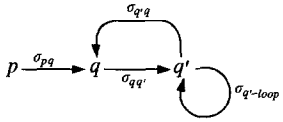
Step 5: Goto 2.

We now observe the following:

- The above "algorithm" terminates: One only expands *active* nodes and since $(V,E)_N$ is finite expansion cannot continue indefinitely because of the annotation of the formulas.
- All claims stated in the algorithm are valid: since the strategy used to compute the tree is to first try to prove that A_2 holds at a state, and if not, then expand the tree, we conclude that:
  - *Claim* 1 *is valid*: If $q \models \neg A_2$ and $q \models \neg A_1$, then because of the way the tree is expanded we could exhibit a finite path from p along which $A_1 \wedge \neg A_2$ holds until

$\neg A_1 \wedge \neg A_2$ holds. But since any finite path can be extended to a computation (K is assumed to be finite) we obtain a contradiction with the assumption $p \models A_1\ U_\forall\ A_2$.

- *Claim* 2 *is valid*: If $next(q) = \emptyset$, then we would have found a finite path starting at p and ending in q, a deadlock. This would be a computation from p to q along which no state satisfied A_2. Again, this would contradict $p \models A_1\ U_\forall\ A_2$.
- *Claim* 3 *is valid*: Since $q \in \mathsf{C}$ we conclude $next(q) \neq \emptyset$.
- *Claim* 4 *is valid*: If $T' = \emptyset$, then because T' keeps track of which transitions have been concurrently enabled along the loop starting and ending at q (along the branch from the root of the tree to the current node), we would have detected one or more loops of the shape



along which A_2 never holds, and by repeating these loops we could exhibit an infinite computation along which A_2 never holds. This contradicts $p \models A_1\ U_\forall\ A_2$.

- *Claims* 5 and 6 *are valid*: As for Claim 2.

Assume the produced tree is not tableau. Then, using the induction hypothesis, we conclude that the only reason why the tree is not a tableau is that it has leaves labelled by terminal sequents. But then an argument similar to that used to show the validity of Claim 4 gives us a contradiction with the assumption $p \models A_1\ U_\forall\ A_2$.

"*If*" *direction* (*soundness*): We show that if there exists a tableau T with root $p \vdash Ann(A_1\ U_\forall\ A_2)$, then $p \models A_1\ U_\forall\ A_2$. So assume that $p \models \neg(A_1\ U_\forall\ A_2)$, i.e., $p \models \neg A_2$ and there exists a $\sigma \in [\sigma'] \in comp(p)$ such that one of the following cases hold:

– $|\sigma| < \omega$, $p = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} \cdots \xrightarrow{t_m} p_m \not\rightarrow$, $\sigma = t_1 \cdots t_m$, and

$$(\forall n \leqslant |\sigma|.\ (p_n \models \neg A_2) \vee (\exists 0 \leqslant i < n.\ p_i \models \neg A_1))$$

There are two cases.

- Assume $(\exists\, 0 < i \leqslant m.\ p_i \models A_2)$. Let $i_0 > 0$ denote the least such index. We know that there must exist an index $0 \leqslant j < i_0$ such that $p_j \models \neg A_1$. Let j_0 denote the least such index. Clearly, the path $t_1 \cdots t_{j_0}$ can be assumed to be loop free and traceable in T along nodes q, such that there exists a tableau with root $q \vdash B_1\ U_\forall\ B_2$ (using the induction hypothesis to obtain contradictions). But this gives a contradiction since T must then have a subtree which is a tableau labelled with root $q_{j_0} \vdash Ann(A_1)$, i.e., $p_{j_0} \models A_1$.
- No states along σ satisfies A_2. If there is a state which satisfies $\neg A_1$ along the path, the argument above can be applied. Else, for any $0 \leqslant i \leqslant m$ we have $p_i \models A_1 \wedge \neg A_2$. But then there must exist a loop free path from p to p_m such that $A_1 \wedge (\neg A_2)$ is satisfied along it and this path must be traceable in T. But this means there must exist a leaf labelled $p_m \vdash Ann(A_1)\ U_\forall^{\mathsf{C}}\ Ann(A_2)$ such that $p_m \notin \mathsf{C}$, and since $p_m \not\rightarrow$, T cannot be a derivation tree.

– $|\sigma| = \omega$, $p = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} \cdots$, $\sigma = t_1 t_2 \cdots$, and

$$(\forall n \in \mathbb{N}.\ (p_n \models \neg A_2) \vee (\exists 0 \leqslant i < n.\ p_i \models \neg A_1))$$

As before, we extract two cases:

- $(\exists i \in \mathbb{N}.\ p_i \models A_2)$. Let $i_0 > 0$ be the least such index. As before we have a least index $0 \leqslant j_0 < i_0$ such that $p_{j_0} \models \neg A_1$. By repeating the above argument, we obtain a contradiction.
- $(\forall n \in \mathbb{N}.\ p_n \models \neg A_2)$. If there is a state which satisfies $\neg A_1$ along the path, the above argument can be applied. Else, we can obtain a path $\sigma' \in [\sigma'] \in comp(p)$ from σ such that $p \xrightarrow{\sigma'} = p'_0 \xrightarrow{t'_1} p'_1 \xrightarrow{t'_2} \cdots \xrightarrow{t'_{m-1}} p'_{m-1} \xrightarrow{t'_m} p'_{k_0} \xrightarrow{t'_{k_0+1}} p'_{k_0+1} \cdots$, where $p'_0, \ldots, p'_{m-1}$ occur only once along σ' while $p''_{k_0}, \ldots$ occur infinitely often along σ'. (We simply remove a finite number of loops in σ, since $(V,E)_N$ is finite.) The common suffix ensures that no transition is cc-enabled along σ'. Since no transition is cc-enabled along σ' there must exist finitely many nonempty loops $\sigma_{loop_1}, \ldots, \sigma_{loop_r}$ starting and ending at p''_{k_0}, such that no transition from $next(p''_{k_0})$ is cc-enabled along $(\sigma_{loop_1} \cdots \sigma_{loop_r})^\omega$, i.e., no enabled transition at p_{k_0} is independent of all transitions taken in the r loops. Notice that these loops might themselves contain loops. Also, since $|next(p''_{k_0})| \leqslant |T|$ we may assume that $1 \leqslant r \leqslant |T|$. Let $next(p''_{k_0}) = \{t'''_{j_1}, \ldots, t'''_{j_r}\}$. We may also assume that σ_{loop_l} corresponds to a loop along which some transition in conflict with t'''_{j_l} is taken.

From each loop σ_{loop_l} we can extract, by deleting inner loops, three paths σ'_{loop_l}, σ''_{loop_l}, and σ'''_{loop_l} such that

(i) σ''_{loop_l} contains a transition in conflict with t'''_{j_l},

(ii) $p''_{k_0} \xrightarrow{\sigma'_{loop_l}} q_{k_0} \xrightarrow{\sigma''_{loop_l}} q_{k_0} \xrightarrow{\sigma'''_{loop_l}} p''_{k_0}$, for some q_{k_0},

(iii) $p''_{k_0} \xrightarrow{\sigma'_{loop_l}} q_{k_0}$ and $q_{k_0} \xrightarrow{\sigma'''_{loop_l}} p''_{k_0}$ are loop free,

(iv) $q_{k_0} \xrightarrow{\sigma''_{loop_l}} q_{k_0}$ is a simple loop,

(v) all states along this new loop satisfy $A_1 \wedge \neg A_2$

But then, using the induction hypothesis, a prefix of the following path must be traceable in the tableau T:

$$p \xrightarrow{t'_1} \cdots \xrightarrow{t'_m} p''_{k_0} \xrightarrow{\sigma} p''_{k_0} \xrightarrow{\sigma'_{loop_1}} \xrightarrow{\sigma''_{loop_1}} \xrightarrow{\sigma'''_{loop_1}} \cdots \xrightarrow{\sigma'_{loop_r}} \xrightarrow{\sigma''_{loop_r}} \xrightarrow{\sigma'''_{loop_r}} p''_{k_0}$$

where σ is a nonempty simple loop obtained by deleting inner loops from the loop $\sigma'_{loop_1}\sigma''_{loop_1}\sigma'''_{loop_1}$ (rule 7 is going to be applied). The path must also end in a leaf labelled $p''_{k_0} \vdash Ann(A_1)\, U_\forall^{(p''_{k_0}, n, \emptyset)}\, Ann(A_2)$, because the rules 9 and 12 keep track (in the annotation) of which transitions have been concurrently enabled. In our case there are no such transitions, so T cannot be a tableau and we obtain the desired contradiction. This completes the proof. □

As an example, we show that the net from Fig. 2 will eventually be able to fire a transition labelled by a c action (assume the transitions are t_1, t_2, and t_3 and are

$$\dfrac{\dfrac{i \vdash tt\, U_{\forall}^{\emptyset}(<c>tt)}{i \vdash tt} \quad \dfrac{\dfrac{i \vdash tt\, U_{\forall}^{\{i\}}(<c>tt)}{\dfrac{i \vdash tt\, U_{\forall}^{(i,2,\{t_1,t_2\})}(<c>tt)}{\mathsf{T}_1}} \quad \dfrac{s_1 \vdash tt\, U_{\forall}^{\{i\}}(<c>tt)}{\dfrac{s_1 \vdash <c>tt}{s_2 \vdash tt}}}$$

where T_1 is

$$\dfrac{\dfrac{\dfrac{\dfrac{i \vdash tt\, U_{\forall}^{(i,1,\{t_1,t_2\},\emptyset,\rightarrow)}(<c>tt)}{\dfrac{i \vdash tt\, U_{\forall}^{(i,1,\{t_2\},\{i\},\rightarrow)}(<c>tt)}{\dfrac{i \vdash tt\, U_{\forall}^{(i,1,\{t_2\},\emptyset,\leftarrow)}(<c>tt)}{i \vdash tt\, U_{\forall}^{(i,1,\{t_2\})}(<c>tt)}}} \quad \dfrac{s_1 \vdash tt\, U_{\forall}^{(i,1,\{t_1\},\{i\},\rightarrow)}(<c>tt)}{\dfrac{s_1 \vdash <c>tt}{s_2 \vdash tt}}}{\mathsf{T}_2} \quad i \vdash tt}$$

where T_2 is

$$\dfrac{\dfrac{i \vdash tt\, U_{\forall}^{(i,0,\{t_2\},\emptyset,\rightarrow)}(<c>tt)}{i \vdash tt} \quad \dfrac{\dfrac{\dfrac{i \vdash tt\, U_{\forall}^{(i,0,\{t_2\},\{i\},\rightarrow)}(<c>tt)}{\dfrac{i \vdash tt\, U_{\forall}^{(i,0,\{t_2\},\emptyset,\leftarrow)}(<c>tt)}{i \vdash tt\, U_{\forall}^{(i,0,\{t_2\})}(<c>tt)}} \quad \dfrac{s_1 \vdash tt\, U_{\forall}^{(i,0,\emptyset,\{i\},\rightarrow)}(<c>tt)}{\dfrac{s_1 \vdash <c>tt}{s_2 \vdash tt}}}$$

Fig. 4. Example of a tableau.

labelled a, b, and c, respectively). By the previous theorem, to show $i \models \mathsf{Ev}(\langle c \rangle tt)$ it is sufficient to construct a tableau with root $i \vdash tt\, U_{\forall}^{\emptyset}(\langle b \rangle tt)$. Fig. 4 shows such a tableau.

Notice that if we restrict ourselves to labelled 1-safe nets where the independence relation is empty and translate $A_1\ U_{\exists}\ A_2$ into $\mu X.A_2 \vee (A_1 \wedge \langle Act \rangle X)$ and $A_1\ U_{\forall}\ A_2$ into $\mu X.A_2 \vee (A_1 \wedge \langle Act \rangle tt \wedge [Act]X)$ (actually applying this translation recursively on the subformulas A_1 and A_2), our proof rules will work in essentially the same manner as those presented in [16, 33].

Choosing an instance of the model-checking problem to be a pair (N, A) consisting of a labelled 1-safe net and a formula A and defining its size to be the sum of the size of the net and the length of the formula, we obtain the following complexity result.

Theorem 18. *The model-checking problem is PSPACE-complete.*

Proof (*sketch*). The hardness result follows from easy modifications of the results in [7], while the PSPACE upper bound can be obtained using the techniques from [5] based on the observations in Section 5.1 (the bound on the number and length of the γ's). □

6. Conclusion

In the context of model-checking, partial order semantics have several advantages. The so-called "state-space explosion" problem has motivated researches to use partial order semantics. It has been observed that an exhaustive state space exploration can often be avoided; e.g., if a sequence (element) of a trace leads to a deadlocked state, then all sequences in that trace must necessarily lead to that deadlock. Hence, it is sufficient only to explore one sequence in that trace. This can lead to significantly improved running times and space consumptions as observed, among others, by Valmari [35, 36] and by Godefroid and Wolper [10, 11, 40, 12].

Another motivation to investigate partial order semantics is that interleaving models of concurrency have failed to provide an acceptable interpretation of what it means for events of a concurrent system to be independent. Partial order semantics allows one to, e.g., interpret temporal logics over traces taking causality and concurrency into account, see [26, 27, 34, 1, 28]. Much work has been devoted to transfer notions and results from the interleaving models to the "true concurrency" models [14, 13, 39, 17]. Trying to contribute to the "transferring of results" we have provided two verification methods for a CTL-like logic interpreted over maximal traces.

An issue not pursued here – and still left open – is a decision procedure for P-CTL. In [6] we examine several variants of P-CTL augmented with operators which directly express the presence or absence of concurrent behaviour. It turns out that a restricted version of the satisfiability problem for these logics is undecidable. Axiomatisations of similar logics have been investigated in, e.g., [17].

Another issue is whether or not it is possible to exploit ordered binary decision diagrams [4] as done in [3] to construct an efficient model-checker for logic.

Acknowledgements

The author would like to thank Mogens Nielsen, Nils Klarlund and the anonymous AMAST'95 and TCS referees for helpful comments on a draft of the paper.

References

[1] R. Alur, D. Peled and W. Penczek, Model-checking of causality properties, in: *Proc. Symp. on Logic in Computer Science* (IEEE, New York, 1995) 90–100.

[2] M.A. Bednarczyk, Categories of asynchronous systems, Ph.D. Thesis, University of Sussex, 1988, Ph.D. in Computer Science, Report no.1/88.

[3] J.R. Bruch, E.M. Clarke, K.L. McMillan, D.L. Dill and L.J. Hwang, Symbolic model checking: 10^{20} states and beyond, *Inform. Comput.* **98** (1992) 142–170.

[4] R.E. Bryant, Graph-based algorithms for boolean function manipulation, *IEEE Trans. Comput.* **35** (1986) 677–691.

[5] A. Cheng, Complexity results for model checking, Research Series RS-95-18, BRICS, Department of Computer Science, University of Aarhus, February 1995.

[6] A. Cheng, Petri nets, traces, and local model checking, Research Series RS-95-39, BRICS, Department of Computer Science, University of Aarhus, July 1995. Full version of paper appearing in *Proc. AMAST'95* (Springer, Berlin, 1995).

[7] A. Cheng, J. Esparza and J. Palsberg, Complexity results for 1-safe nets, *Theoret. Comput. Sci.* **147** (1995) 117–136.

[8] E.M. Clarke, E.A. Emerson and A.P. Sistla, Automatic verification of finite state concurrent system using temporal logic, *ACM Trans. Programming Languages Systems* **8** (1986) 244–263.

[9] V. Diekert and G. Rozenberg, *The Book of Traces* (World Scientific, Singapore, 1995).

[10] P. Godefroid, Using partial orders to improve automatic verification methods, in: *Proc. CAV'90, 2nd Workshop on Computer-Aided Verification*, Lecture Notes in Computer Science, Vol. 531 (Springer, Berlin, 1990) 176–185.

[11] P. Godefroid and P. Wolper, Using partial orders for the efficient verification of deadlock freedom and safety properties, in: K.G. Larsen and A. Skou, eds., *Proc. CAV'91, Computer-Aided Verification*, 575 (Springer, Berlin, 1991) 332–343.

[12] P. Godefroid and P. Wolper, A partial approach to model checking, *Inform. Comput.* **110** (1994) 305–326.

[13] L. Jategaonkar and A. Meyer, Deciding true concurrency equivalences on finite safe nets, in: *Proc. ICALP'93* (1993) 519–531.

[14] A. Joyal, M. Nielsen and G. Winskel, Bisimulation and open maps, In: *Proc. LICS'93, 8th Ann. Symp. on Logic in Computer Science* (1993) 418–427.

[15] M.Z. Kwiatkowska, Event fairness and non-interleaving concurrency, *Formal Aspects Comput.* **1** (1989) 213–228.

[16] K.G. Larsen, Proof systems for Hennessy–Milner logic with recursion, in: *Proc. CAAP*, Nancy, France, Lecture Notes in Computer Science, Vol. 299 (Springer, Berlin, 1988) 215–230.

[17] K. Lodaya, R. Parikh, R. Ramanujam and P.S. Thiagarajan, A logical study of distribution transition system, *Inform. Comput.* **119** (1995) 91–118.

[18] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems* (Springer, Berlin, 1992).

[19] A. Mazurkiewicz, Trace theory, in: *Petri Nets: Applications and Relationships to Other Models of Concurrency*, Lecture Notes in Computer Science, Vol. 255 (Springer, Berlin, 1986) 279–324.

[20] A. Mazurkiewicz, Introduction to trace theory, in: V. Diekert and G. Rozenberg, eds., *The book of traces* (World Scientific, Singapore, 1995) 3–41.

[21] A. Mazurkiewicz, E. Ochmański and W. Penczek, Concurrent systems and inevitability, *Theoret. Comput. Sci.* **64** (1989) 281–304.

[22] R. Milner, in: C.A.R. Hoare, *Communication and Concurrency*, Prentice Hall International Series in Computer Science (Prentice-Hall, Englewood Cliffs, NJ, 1989).

[23] M. Mukund and M. Nielsen, CCS, locations and asynchronous transition systems, in: *Proc. Foundations of Software Technology and Theoretical Computer Science*, Vol. 12 (1992) 328–341.

[24] M. Nielsen, G. Plotkin and G. Winskel, Petri nets, event structures and domains, Part I *Theoret. Comput. Sci.* **13** (1981) 85–108.

[25] E.R. Oldeog, *Nets, Terms and Formulas, Tracts in Theoretical Computer Science*, Vol. 23 (Cambridge Univ. Press, Cambridge, 1991).

[26] D. Peled and A. Pnueli, Proving partial order liveness properties, in: *Proc ICALP'90*, Lecture Notes in Computer Science, Vol. 443 (Springer, Berlin, 1990) 553–571.

[27] W. Penzcek, Temporal logics for trace systems: on automated verification, *Internat. J. Foundations Comput. Sci.* **4** (1993) 31–67

[28] W. Penczek and R. Kuiper, The book of traces, in: V. Diekert and G. Rozenberg, eds., Traces and Logic, Ch. 10. (World Scientific, Singapore, 1995) 307–381.

[29] C.A. Petri, Kommunikation mit automaten, Schriften des iim nr. 2, Bonn, Institut für Instrumentelle Mathematik, 1962, Also in: New York: Griffiss Air Force Base, Tech. Report RADC-TR-65-377, Vol. 1 Suppl. 1, 1966. English translation.

[30] W. Reisig, *Petri Nets – An Introduction*, EATCS Monographs in Computer Science, Vol. 4, 1985.

[31] M.W. Shields, Concurrent machines, *Comput. J.* **28** (1985) 449–465.

[32] E.W. Stark, Concurrent transition systems, *Theoret. Comput. Sci.* **64** (1989) 221–269.

[33] C.P. Stirling and D. Walker, Local model checking in the modal mu-calculus, Tech. Report ECS-LFCS-89-78, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, May 1989.

[34] P.S. Thiagarajan, A trace based extension of linear time temporal logic, in: *Proc. Symp. on Logics in Computer Science* (IEEE, New York, 1994).

[35] A. Valmari, Error detection by reduced reachability graph generation, in: *Proc. 9th European Workshop on Application and Theory of Petri Nets* (Venice, Italy) (1988) 95–112.

[36] A. Valmari, Stubborn sets for reduced state space generation, in: G. Rozenberg, ed., *Advances in Petri Nets*, Lectures Notes in Computer Science, Vol. 483 (Springer, Berlin, 1990) 491–515.

[37] G. Winskel, Events in computation, Ph.D. Thesis, University of Edinburgh, 1980.

[38] G. Winskel, Event structures, in: *Petri Nets: Applications and Relationships to Other Models of Concurrency*, Lecture Notes in Computer Science, Vol. 255 (Springer, Berlin, 1986)

[39] G. Winskel and M. Nielsen, Models for concurrency in: S. Abramsky, D.M. Gabbay and T.S.E. Gabbay, eds. *Handbook of Logic and the Foundations of Computer Science*, Vol. 4 (Oxford University Press, Oxford 1995)

[40] P. Wolper and P. Godefroid, Partial-order methods for temporal verification, Tech. Report, Université de Liége, Institut Montefiore, August 1993. Wolper-1, Hand-outs at Summerschool in Logical Methods in Concurrency Aarhus'93; *Concur'93 Proc.*, to appear.